FIR FILTER HARDWARE IMPLEMENTATION

Report for Module 5M01 Integrated System Design

Lingyu Gong, Student ID 23337765
Trinity College Dublin
gongl@tcd.ie

November 2023

This report is submitted in part fulfilment of the assessment required in EE5M01 Integrated System Design. I have read and understand the plagiarism provisions in the General Regulations of the University Calendar for the current year. These are found in Parts II and III at http://www.tcd.ie/calendar.

The same FIR filter is investigated in this experiment, but it is based on the previous experiment III continuing the exploration towards the hardware. In the end, we were able to implement the filter in hardware by linking it to the PYNQ Z2 hardware with the help of one of the IP cores. At the same time, the board PYNQ Z2 can be connected to Jupyter Notebook and further realized by Python programming.

1 Design Filter

First, a brief introduction to several common filters:

1. Low-pass filter: The inductor prevents high-frequency signals from passing through and allows low-frequency signals to pass through. The characteristics of the capacitor are opposite. A filter that allows signals to pass through an inductor or a filter that is connected to the ground through a capacitor has less attenuation of low-frequency signals than high-frequency signals and is called a low-pass filter.

2. High-pass filter: The simplest high-pass filter is the "first-order high-pass filter". Its characteristics are generally expressed by a first-order linear differential equation. Its left side is

exactly the same as the first-order low-pass filter, only the right side is Derivatives of the excitation source rather than the excitation source itself. When lower frequencies pass through the system, there is little or no output; when higher frequencies pass through the system, there will be less attenuation.

3. Bandpass filter: A circuit that only allows specific frequencies to pass while effectively suppressing signals at other frequencies. A filter that can pass frequency components in a certain frequency range but attenuate frequency components in other ranges to very low levels, as opposed to the concept of a band-stop filter.

2 Overlay

Our current system utilizes the Zynq z2 board, a sophisticated System on a Chip (SOC) that merges a dual-core processor (PS) with an FPGA structure (PL). The PS system has a range of distinctive peripherals, which can be broadened through the PL system. The FPGA is precisely customized in the overlay or hardware library to enhance the user applications from the Zynq processing system to the programmable logic. By using PYNQ, a Python interface, it is simple to control the overlays in the PL from Python operating in PS, providing efficient PL management from the PS.

2.1 Basic Introduction

An FPGA overlay, at its core, functions as a specialized hardware library. It merges a generic FPGA design with a software abstraction layer that provides a well-defined API. The PYNQ-Z2 board is notable for its base overlay, facilitating Python access to both internal and external peripherals. Illustrated in the diagram below1, this overlay simplifies the interaction between Python scripts and the FPGA hardware. Python's high-level characteristics enhance accessibility, allowing a diverse range of developers to harness FPGA capabilities for tasks like real-time processing, accelerated computations, and interfacing with peripherals. The PYNQ-Z2's base overlay acts as a foundational framework, making FPGA-based acceleration and customization more approachable for a broader developer community. Cited from PYNQ Introduction

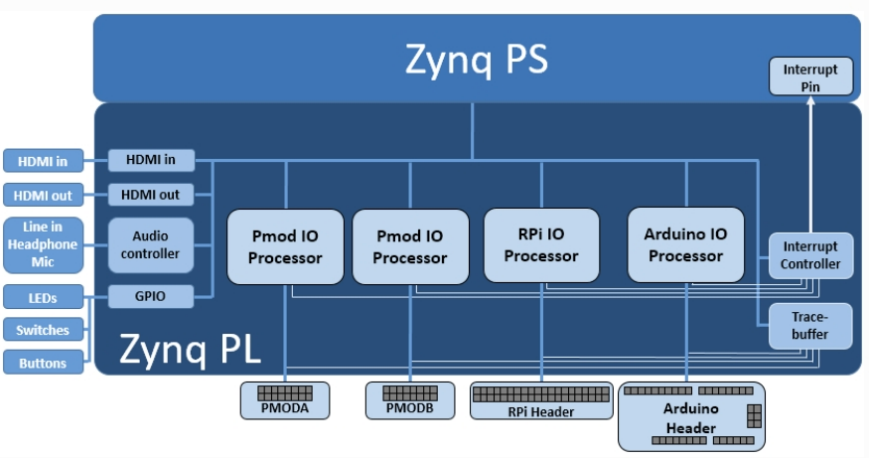

Figure 1: *PYNQ-Z2 Block Diagram.*

2.2 IP cores

There are a total of 17 IP cores that have been generated and they are displayed in the figure below2.

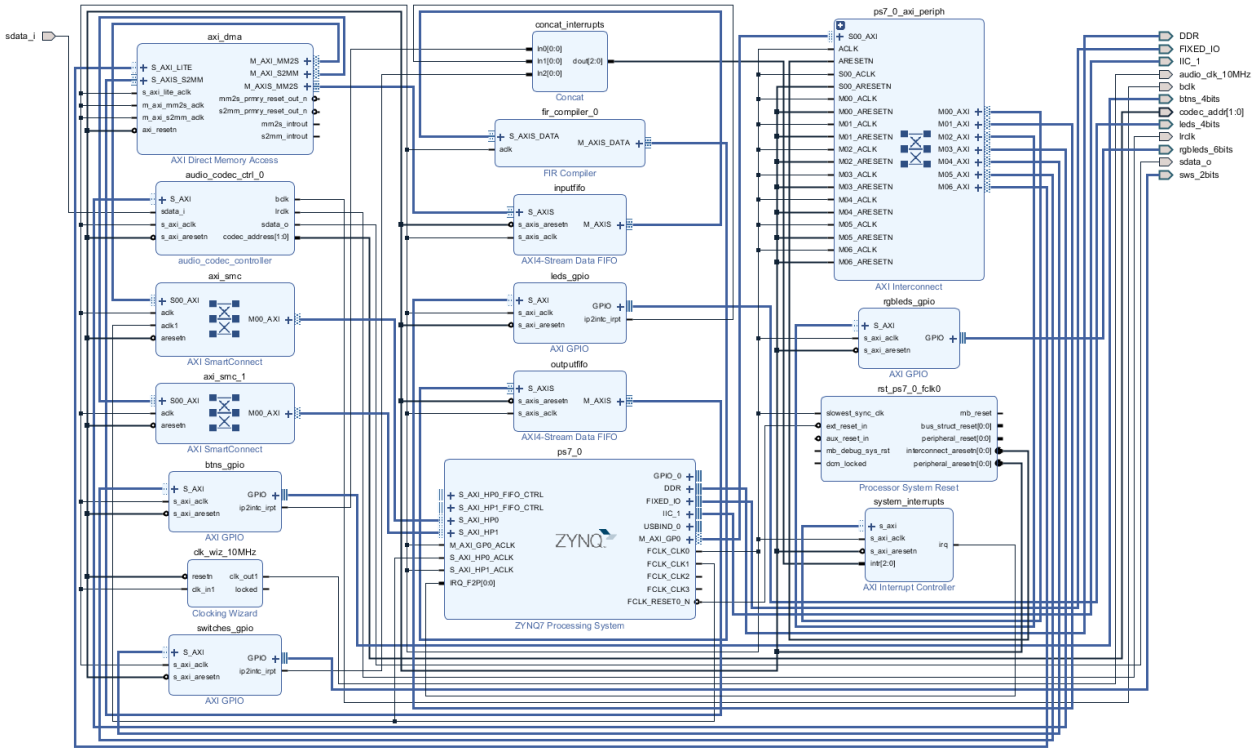

Figure 2: *IP cores generated by VIVADO*

To be more specific, I make a summary below.3.

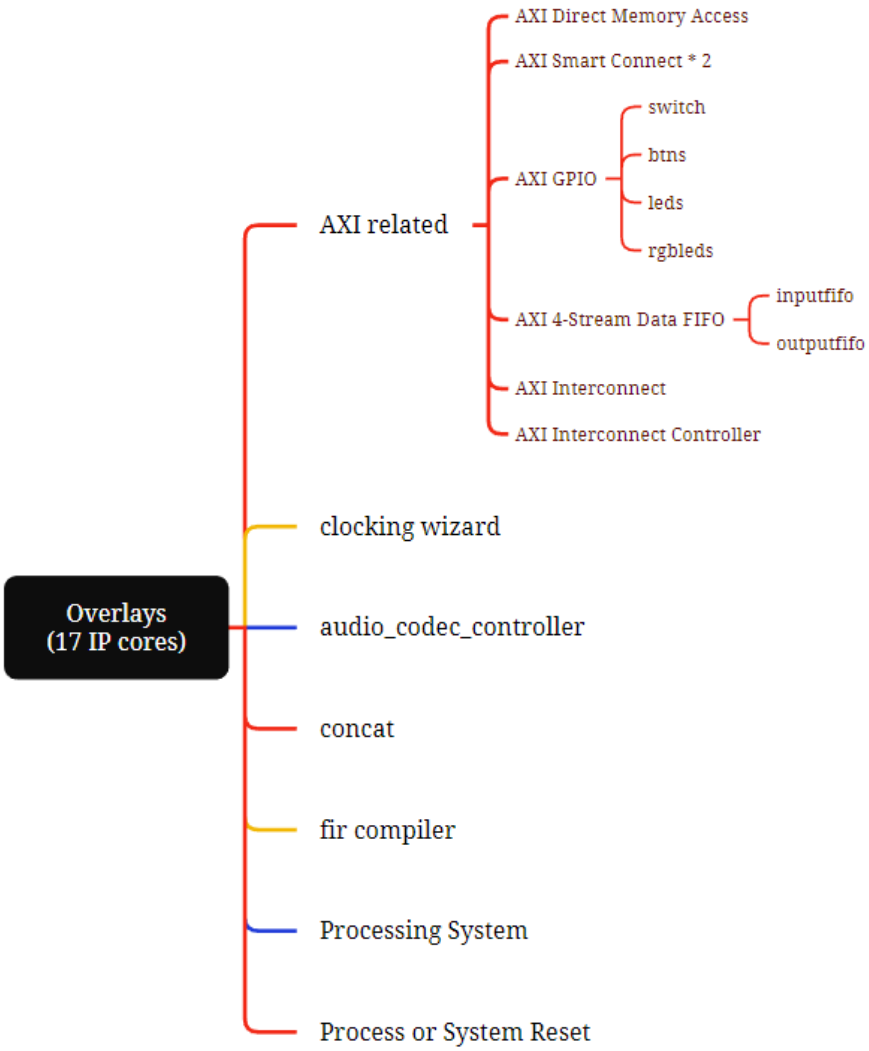

Figure 3: *IP cores in details*

3 Make Comparison

The first graph shows the differences after filtering the degraded audio in the software4.

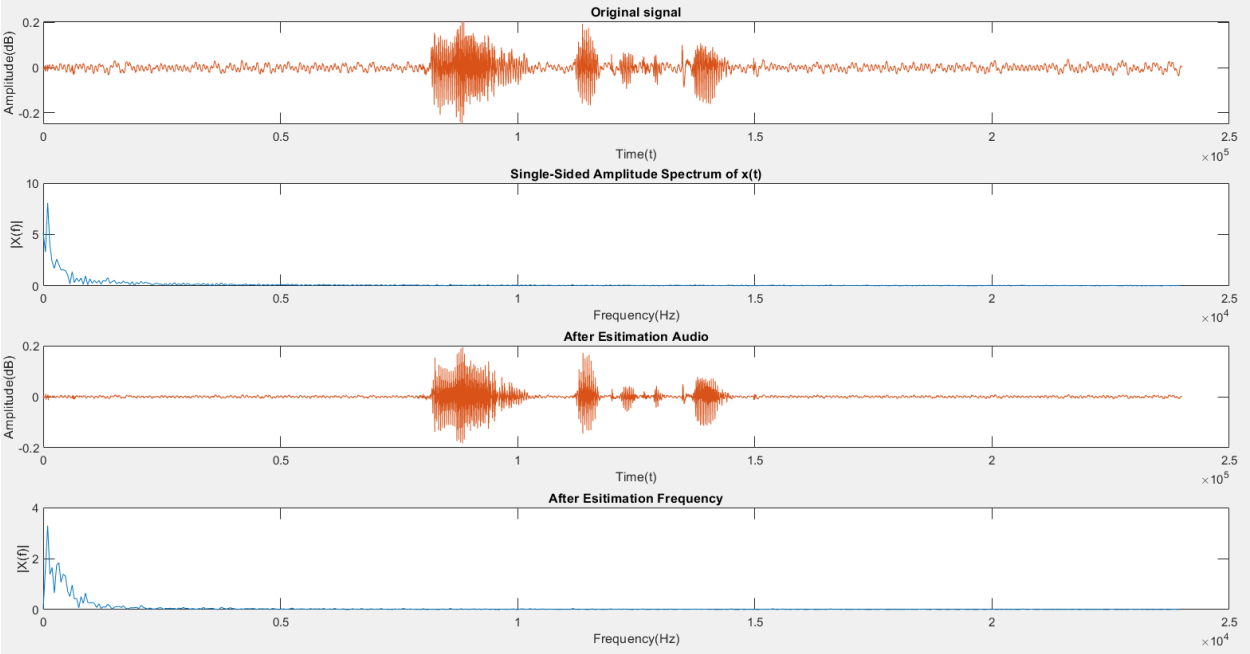

Figure 4: *Compare the audio before and after in software*

The second graph shows the differences after filtering the degraded audio in the hardware5.

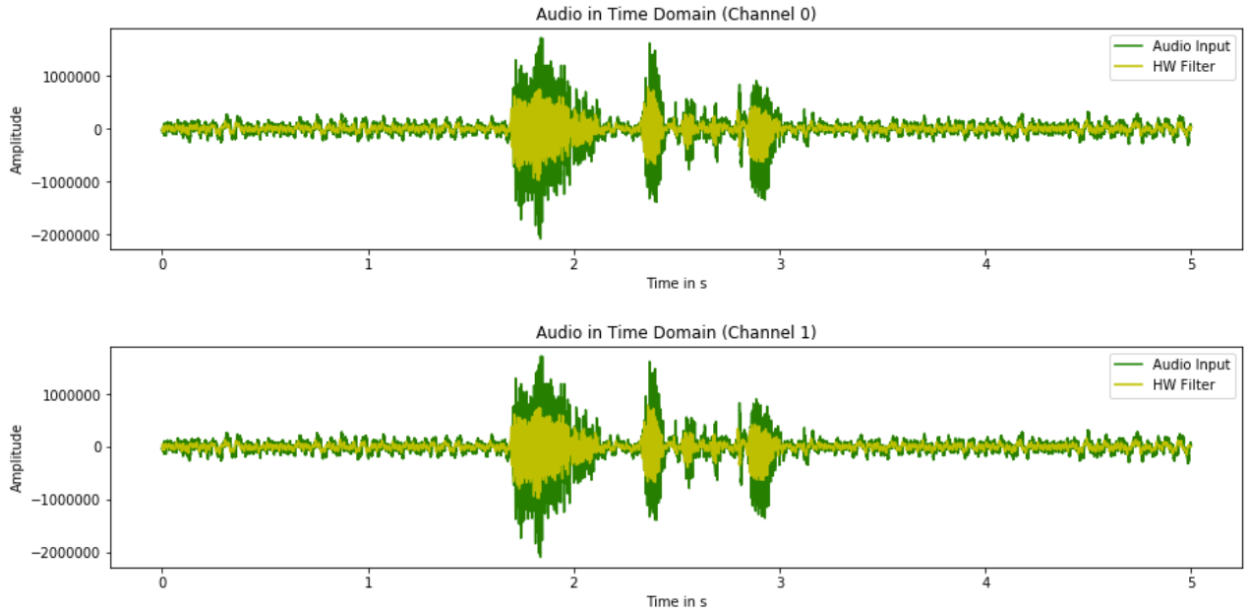

Figure 5: *Compare the audio before and after in hardware*

4 Conclusions

In summary, There are two ways to filter noise: hardware filtering and software filtering. Hardware filtering uses analogue filters, such as RC filters which consist of resistors and capacitors with specific specifications, or LC filters which consist of inductors and capacitors. On the other hand, software filtering, also known as digital filtering, reduces or weakens noise by performing certain calculations or judgment procedures.

4.1 Hardware

Implementing hardware filtering involves the usage of a dedicated filter circuit that possesses the capability of processing signals instantly and providing real-time performance. The first key benefit of hardware filtering is that it processes the signal without any additional computational delay right after the signal is input. The second key benefit is that it is performed directly in the circuit, resulting in real-time processing and making it the suitable choice for scenarios that call for high levels of real-time performance.

4.2 Software

Signal filtering is the process of removing unwanted components from a signal. Digital signal processing algorithms are used to implement this process through software filtering. One of the key advantages of software filtering is its high flexibility due to its discrete signal processing characteristics. Since software filtering is processed in the discrete-time domain, continuous signals need to be converted into discrete signals for processing. Additionally, software filters can be selected based on specific needs. This allows for the use of different filtering algorithms and parameters, providing high flexibility and adjustability.

4.3 Trade-off

Hardware filtering is ideal for applications that require high real-time performance and use relatively simple filtering functions such as audio processing and real-time data collection. On the other hand, software filtering is more appropriate for applications that require complex filtering algorithms and flexible parameter adjustments, such as image processing and signal analysis.